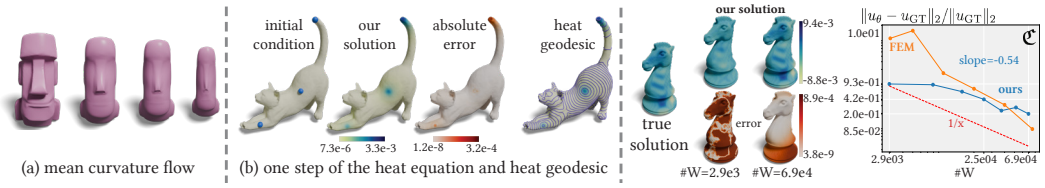
PINNSUR: PHYSICS-INFORMED NEURAL NETWORKS FOR PDES ON CURVED SURFACES

Anonymous authors
Paper under double-blind review



(a) mean curvature flow

(b) one step of the heat equation and heat geodesic

Figure 1: PINNsur solves PDEs on curved surfaces embedded in 3D. (a) Our method simulates a mean curvature flow on a mesh. (b) We solve a step of the heat equation given initial heat sources and solve for the heat geodesic distance. (c) We then show how the error in the predicted solution converges as the network weights ($\#W$) are increased. The ground truth is the cotangent Laplacian solution on a very fine mesh.

ABSTRACT

Partial differential equations (PDEs) on surfaces are fundamental to scientific computing and geometry processing. A popular approach to solving PDEs on surfaces is the finite element method (FEM), where the surface is divided into discrete geometric elements (usually triangles). Recently, physics-informed neural networks (PINNs) have emerged as a continuous, mesh-free alternative that does not suffer from FEM's sensitivity to mesh quality or geometric discretization errors. We present PINNsur, a simple framework for using PINNs on curved surfaces: we train a neural field to approximate the surface's normals, and then we express surface differential operators using their projection from $\mathbb{R}^3$ onto the surface. Since every orientable smooth manifold has well-defined normals, our method is suitable for all such surfaces, regardless of curvature or topology, enabling many geometry processing applications. Moreover, despite their empirical success in solving PDEs in flat Euclidean domains, PINNs lack convergence guarantees. This gap restricts their adoption as reliable solvers compared to established methods like FEM, where convergence to the true solution is well understood and theoretically grounded. Surface PDEs are particularly challenging to solve convergently, as one must not only deal with the convergence of the function approximation, but also with the convergence of the geometric approximation of the surface itself. In this work, we empirically show that our formulation to solve surface PDEs results in a convergent behavior with respect to the number of tunable weights in the network, and analyze the behavior of other PINNs using this criterion.

1 INTRODUCTION

Partial Differential Equations (PDEs) on curved two-dimensional surfaces $\Omega \subseteq \mathbb{R}^3$ are at the core of many 3D geometry processing applications such as surface editing, animation, physical simulation, and computer-aided manufacturing. Consequently, accurate, efficient, and easy-to-implement solvers for surface PDEs have become a central component of modern geometry processing pipelines. Curved surfaces pose additional challenges to PDE solving compared to flat two-dimensional regions $\subseteq \mathbb{R}^2$ or planar subsets in $\mathbb{R}^3$: One must not only discretize the PDE, but also the surface itself. An extremely common discretization of surfaces has been the triangle mesh, which works well together with the Finite Element Method (FEM) for discretizing PDEs, especially linear (Lagrangian) FEM, which approximates solutions using piecewise linear functions defined on triangular elements. Linear FEM has many advantages, the main one being that it can solve the Poisson equation in a convergent way given certain conditions (Dziuk, 1988; Wardetzky, 2007). It also has a variety of drawbacks, such as needing meshes with well-behaved triangles and careful refinement.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To address FEM's drawbacks, Physics-Informed Neural Networks (PINNs) have become popular (Karniadakis et al., 2021; Hao et al., 2022), where the function is approximated by a differentiable neural network, and the PDE is solved using *autodifferentiation*. Neural network training methods are then used to find the weights that solve the PDE. PINNs address some of FEM's shortcomings, such as the dependence on a mesh with high-quality triangles, but are difficult to extend to curved surfaces: The coordinate networks underlying PINNs generally only work on d-dimensional flat subsets of $\mathbb{R}^d$ (i.e., areas in $\mathbb{R}^2$ and volumes in $\mathbb{R}^3$). Recent works (Williamson & Mitra, 2025; Welschinger et al., 2025) have begun to extend PINNs to curved surfaces and work towards integrating PINNs in mainstream geometry processing applications. One line of work formulates PINNs in $\mathbb{R}^3$, but explicitly evaluates a surface differential operator (such as the Laplace-Beltrami operator Δ) using standard formulas from differential geometry. Hu et al. (2024) use a level-set representation to obtain the surface normal and mean curvature and rewrite the Laplace–Beltrami operator using Cartesian differential operators. In contrast, Williamson & Mitra (2025) parameterize the surface using a spherical neural map and construct the Laplace–Beltrami operator from that parameterization. There is no explicit discretization of the PDE or the surface geometry into elements; instead, the effective discretization is implicitly controlled by the number of network weights. Thus, unlike FEMs, which provide convergence guarantees to the true solution under consistent discretization (typically as a function of the number of triangular elements n), neural solvers lack analogous guarantees, and systematic studies remain limited.

Building on the ideas of Williamson & Mitra (2025), we introduce a new framework for using PINNs on curved surfaces by completely eliminating the parameter domain, simplifying the network, and extending it to arbitrary manifold topologies. Our key observation is that common surface PDE operators (e.g., Poisson and Helmholtz) do not require a parametric mapping of the surface; normals alone suffice to construct surface differential operators, since one can define the operators in Euclidean space and project out normal components to obtain the surface operators. Our method supports both Dirichlet and Neumann boundary conditions and performs well on many challenging geometries. We then study convergence behavior to the true solution of our proposed surface PDE solver. We introduce a notion for the convergence of PINNs by relating the error to the number of degrees of freedom in the PINN. Our convergence criterion parallels classical convergence criteria in use for FEM, where the error is bounded by an expression containing the number of elements or the element size (both proxies for the total number of degrees of freedom). Analogously, for PINNs, the number of trainable parameters can be interpreted as an effective measure of degrees of freedom. We study both classical PINNs as well as our new PINN for curved surfaces using this criterion, and study their convergence behavior.

Our contributions are:

1. A simple surface PINN framework called PINNsur for solving PDEs using a normals-only neural field for surface operators.
2. An empirical convergence study of PINNs on test functions across varied PDEs, geometries, and boundary conditions.

We refer to Section F for details about the released code and the convergence study on test functions.

2 Related Work

Classical Methods for Surface PDEs. There are many classical methods for solving surface PDEs; we highlight only a few important examples that are used in the geometry processing of curved surfaces. Dziuk (1988) proved an early convergence result for the linear FEM on curved surfaces approximated by triangle meshes, which was later extended by Wardetzky (2007). These results generalize known flat FEM theorems (Brenner & Scott, 2008). As a rule of thumb, such FEM methods have L^2 convergence on the order of $\mathcal{O}(1/n)$, where n is the number of degrees of freedom. The reliability of linear FEM has led to its popularity in many geometry processing applications such as the computation of minimal surfaces (Pinkall & Polthier, 1993), surface fairing (Desbrun et al., 1999), mesh editing (Sorkine et al., 2004), animation (Baran & Popović, 2007), and many more (Botsch et al., 2010). Beyond linear FEM, other popular discretization methods include higher-order FEM (Ferguson et al., 2023), discrete exterior calculus (Hirani, 2003), and Monte Carlo methods (which have only recently been extended to surfaces) (Sugimoto et al., 2024) (which converge on the order of $\mathcal{O}(n^{-1/2})$, where n is the number of degrees of freedom).

Physics Informed Neural Networks. PINNs are neural fields that model solutions to differential equations by posing them as optimization problems supervised by physical laws (Karniadakis et al., 2021; Hao et al., 2022; Raissi et al., 2019; Kharazmi et al., 2021). They offer a flexible alternative to classical discretization-based solvers, especially when data or meshing is challenging. PINNs have been applied across a range of problems, including fluid dynamics (Cai et al., 2021; Jain et al., 2024), volumetric elastodynamics (Rao et al., 2021), Eikonal equations (Smith et al., 2021), solar surface flux transport modeling (Athalathil et al., 2024; 2026) and cloth simulation (Kairanda et al., 2024; 2026). Beyond direct neural solvers, there is growing interest in *neural operators* that learn solution maps for parametric PDEs (Yu et al., 2018; Welschinger et al., 2025). Early PINN works with coordinate-based MLPs struggled with high-frequency signals and derivatives; Siren (Sitzmann et al., 2020) mitigates this via periodic activations and has enabled broader applications (Chen et al., 2023; Zehnder et al., 2021; Kairanda et al., 2024). However, Siren and most PINNs are formulated on Euclidean grids rather than surfaces. Recent work has also applied PINNs to PDEs on general Riemannian manifolds: Ko & Lee (2026) use PINNs to approximate heat kernels for Riemannian diffusion models. Their formulation expresses manifold differential operators using ambient coordinates and tangent-space projections obtained from an explicit specification of the manifold. In contrast, our formulation targets PDEs on embedded surfaces and constructs the required surface differential operators directly from the surface normal field. Recent work by Welschinger et al. (2025) solves PDEs on surfaces via a *neural operator*, but still uses Euclidean finite differences. In contrast, our solver models surface operators directly and solves the PDE on the continuous surface field.

Previous work on PINN convergence includes Shin et al. (2020), who prove that, under certain conditions, PINNs on flat domains in $\mathbb{R}^d$ converge to their solutions, and provide experimental evidence that increasing the amount of training data decreases error. Tang & Fu (2021) investigate the convergence behavior of PINNs on curved surfaces as the number of training samples increases, and they do so for different network depths. Nam et al. (2024) introduce a novel neural Monte Carlo method and investigates its convergence behavior as the network is trained over time. Prior work focuses on whether the training of PINNs converges, while we study convergence to the true solution with respect to degrees of freedom (number of weights), similar to how FEM measures the number of elements.

Neural Geometry Processing. Many recent methods encode geometric shapes (e.g., surfaces) as neural networks, often as neural fields, enabling geometry processing tasks (Yang et al., 2021; Bednarik et al., 2020; Mehta et al., 2022; Gao et al., 2024; Edavamadathil Sivaram et al., 2024; Fargion & Weber, 2025; Chetan et al., 2025). A key advantage is that differential geometry quantities like normals and curvatures can be computed via *autodiff*. Morreale et al. (2021) learn explicit surface maps for distortion-minimizing correspondences, and Morreale et al. (2022) disentangles fine detail from global structure, while Aigerman et al. (2022) learns piecewise-linear mesh mappings. Novello et al. (2022; 2023) explore differentiable geometry of level-set surfaces and how to evolve implicit level-set functions, while Yang et al. (2021) demonstrates surface smoothing and sharpening on implicits. Recently, Chang et al. (2025); Williamson & Mitra (2025) solve the Laplace–Beltrami operator to perform spectral analysis. The closest work to ours is Williamson & Mitra (2025); the key difference is that we avoid a parameterized surface map, since it is not part of the surface PDE operator. We instead use a normal field, which improves accuracy and convergence.

3 METHOD

Our setting is a (curved) two-dimensional manifold surface $\Omega \subseteq \mathbb{R}^3$. We solve PDEs of the form

$$\mathcal{F}[u(\mathbf{x})] = 0, \quad \mathbf{x} \in \Omega, \tag{1}$$

$$u(\mathbf{x}) = g(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega_D, \tag{2}$$

$$\frac{\partial u}{\partial \nu}(\mathbf{x}) = h(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega_N, \tag{3}$$

where $\mathcal{F}$ is a second-order elliptic differential operator, $\partial\Omega_D$ is the boundary curve with Dirichlet condition, and $\partial\Omega_N$ with Neumann condition (where ν is the boundary normal). The no-boundary case $\partial\Omega_D = \partial\Omega_N = \varnothing$ occurs for some curved surfaces. We focus on the Poisson and Helmholtz equations because of their wide applications (Kazhdan & Hoppe, 2013; Pinkall & Polthier, 1993).

3.1 POISSON EQUATION

The (Euclidean) Laplacian $\Delta_{\mathbb{R}^d}$ of a scalar field $u : \mathbb{R}^d \to \mathbb{R}$ ($d = 2, 3$) is defined as the divergence of the gradient of the field. The Laplace-Beltrami operator, or surface Laplacian, is the generalization of the Laplacian $\Delta_{\mathbb{R}^d} u$ to curved domains Ω, $\Delta_\Omega u = \nabla_\Omega \cdot \nabla_\Omega u$. The surface gradient is the orthographic projection of the Euclidean gradient to the local tangent plane $\nabla_\Omega u = \nabla_{\mathbb{R}^d} u - (\nabla_{\mathbb{R}^d} u \cdot \mathbf{n})\mathbf{n}$. Similarly, for a vector field $\mathbf{F} : \mathbb{R}^d \to \mathbb{R}^d$, the surface divergence is defined as

$$\nabla_\Omega \cdot \mathbf{F} = \nabla_{\mathbb{R}^d} \cdot \mathbf{F} - \mathbf{n}^\mathbf{T} \mathbf{J}(\mathbf{F})\mathbf{n}, \text{ where } \mathbf{J}(\mathbf{F}) = \begin{bmatrix} \frac{\partial \mathbf{F}}{\partial x_1} & \frac{\partial \mathbf{F}}{\partial x_2} \cdots \frac{\partial \mathbf{F}}{\partial x_d} \end{bmatrix} \tag{4}$$

is the Jacobian of $\mathbf{F}$ (Hubbard & Burke Hubbard, 2015).

The Poisson equation on a curved domain $\Omega \subset \mathbb{R}^d$ for a scalar function $u : \Omega \to \mathbb{R}$ is given by $\Delta_\Omega u = f$, where $f : \Omega \to \mathbb{R}$ is the source term. It is relatively straightforward to compute the Euclidean Laplacian of a coordinate network by applying *autodifferentiation* to a smooth network $u : \mathbb{R}^d \to \mathbb{R}$:

$$\Delta_{\mathbb{R}^d} u = \frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} + \cdots + \frac{\partial^2 u}{\partial x_d^2} \,. \tag{5}$$

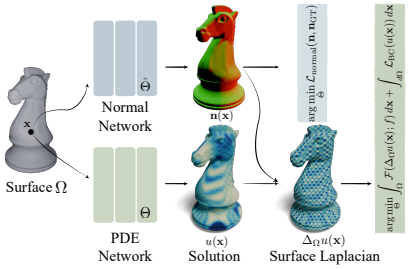


Figure 2: **Overview**. To solve PDEs on a curved surface, we first train a normal network to approximate the surface normals. The PDE network then leverages these normals to compute the surface Laplacian $\Delta_\Omega u$, with boundary conditions added as soft constraints.

This formula does not apply to the Laplace-Beltrami operator, where we can not simply employ component-wise differentiation. Williamson & Mitra (2025) approach this by representing the surface with a parametric neural map $\varphi_\theta : \mathbb{S}^2 \to \Omega$. φ_θ is obtained by first computing a discrete parametrization of the surface triangle mesh from a reference sphere, and then training a neural network to approximate that discrete parametrization. While this allows computing normals via the Jacobian of φ_θ, it does not handle arbitrary topologies (the topology of the parameter domain must match the topology of Ω). In contrast, we present a method that does not require a parametrization at all. Similar to Williamson & Mitra (2025), we also represent u as a coordinate network, but instead of a parametrization we directly work with the normal field that is either known exactly, or that we represent with another network that can be trained quickly and used without any autodifferentiation. The resulting formula for the Laplace-Beltrami operator of a coordinate network is:

$$\Delta_\Omega u = \nabla_{\mathbb{R}^d} \cdot (\nabla_\Omega u) - \mathbf{n}^\top \mathbf{J}(\nabla_\Omega u)\mathbf{n} \,. \tag{6}$$

The solution of the actual PDE works as follows. Given a function f and a surface Ω, we train a PINN to learn the function u by defining the loss function $\mathcal{L}_{\text{poisson}}$ as

$$\mathcal{L}_{\text{poisson}} = \|\Delta_\Omega u_\theta - f\|_2^2. \tag{7}$$

The Laplace-Beltrami operator (Δ_Ω) is computed using equation 6. Please see Fig. 2 for an overview.

3.2 HELMHOLTZ EQUATION

The Helmholtz equation on a curved domain $\Omega \subset \mathbb{R}^d$ for a scalar function $u : \Omega \to \mathbb{R}$ is defined as $\Delta_\Omega u + k^2 u = f$, where k is the wavenumber and $f : \Omega \to \mathbb{R}$ is the source term.

We use the same framework as for the Poisson equation, but define the loss function $\mathcal{L}_{\text{helmholtz}}$ as

$$\mathcal{L}_{\text{helmholtz}} = \|\Delta_\Omega u_\theta + k^2 u_\theta - f\|_2^2. \tag{8}$$

3.3 BOUNDARY CONDITIONS

We employ soft boundary constraints (Raissi et al., 2019; Chen et al., 2023), where boundary conditions are enforced by adding penalty terms to the loss function during training.

$$\mathcal{L}_{\text{soft}} = \mathcal{L}_{\text{PDE}} + \lambda_{\text{BC}} \mathcal{L}_{\text{BC}}, \tag{9}$$

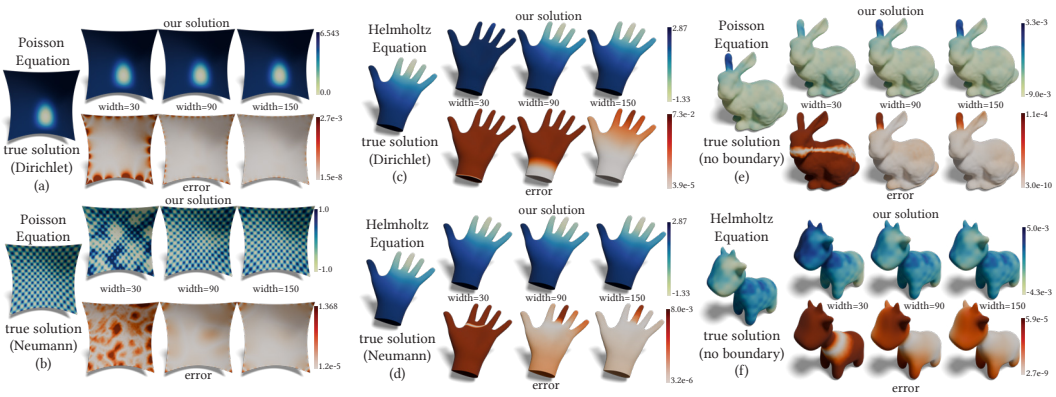


Figure 3: Figure showing the solution and error as network width increases for Poisson and Helmholtz equations under various boundary conditions. We observe that error decreases with increasing width.

where λ_{BC} is a tunable parameter. We choose $\lambda_{\text{BC}} = 100$ to give a high weight to the boundary conditions. Fig. 3 shows that the solution error at the boundary is less dependent on the parameter λ_{BC}, but the error at the boundary decreases as the size of the network increases. Different boundary conditions necessitate different loss functions $\mathcal{L}_{\text{BC}}$.

Dirichlet Boundary. To solve a PDE with Dirichlet boundary condition $u(\mathbf{x}) = g(\mathbf{x}), \mathbf{x} \in \partial\Omega_D$, we add the following term to the loss function.

$$\mathcal{L}_{\text{BC}} = \|u_\theta - g\|_2^2. \tag{10}$$

The loss in equation 10 is minimized exactly when the Dirichlet boundary conditions hold. Since a valid solution to a PDE with Dirichlet boundary conditions exist, there must be a global minimum that both minimizes the main PDE loss (equation 7, equation 8), as well as the additional boundary loss equation 10.

Neumann Boundary. To enforce Neumann boundary conditions, one needs to compute $\frac{\partial u}{\partial \nu}$. Since we enforce Neumann conditions with a soft loss term involving $\frac{\partial u}{\partial \nu}$, this quantity needs to be differentiable, which we achieve by training another coordinate MLP on the boundary normals ν by computing the cross product between the boundary edge and the surface normal at the boundary. Then, we use this MLP to compute the following term at $\partial\Omega_N$

$$\mathcal{L}_{\text{BC}} = \left\|\nabla_\Omega u_\theta \cdot \nu_{\tilde{\theta}} - h\right\|_2^2. \tag{11}$$

Compatibility. When solving the Poisson or Helmholtz equations, the compatibility condition is required to ensure the existence of a solution (Evans, 2010). Existence and uniqueness are guaranteed for both Poisson and Helmholtz when pure Dirichlet boundary conditions are imposed (as long as the boundary condition is compatible with the interior function values). For the Poisson equation $\Delta_\Omega u = f$ and the Helmholtz equation $\Delta_\Omega u + k^2 u = f$ with a Neumann boundary, the solution exists and is unique up to a constant if

$$\int_\Omega f + \int_{\partial\Omega} \frac{\partial u}{\partial \nu} = 0. \tag{12}$$

For surfaces without a boundary, we simply have to ensure $\int_\Omega f = 0$. For all our experiments on surfaces with Neumann or no boundary, we choose source terms f that ensure homogeneous Neumann boundary ($h(\mathbf{x}) = 0$) and $\int_\Omega f = 0$.

3.4 Network Design & Training

Normal Networks. To compute the Laplace-Beltrami operator equation 6, we need the surface normals $\mathbf{n}$ at each point on the surface Ω and boundary normals ν for Neumann boundary conditions. We use a *Siren* network (Sitzmann et al., 2020) which is a multilayer perceptron with $sine$ activation

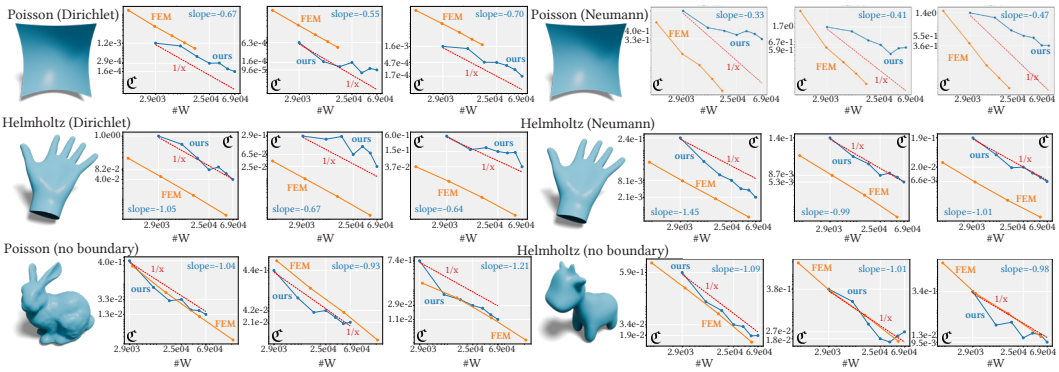


Figure 4: Convergence plots for the Poisson and Helmholtz equations with different boundary conditions for three different functions defined on each surface. In all shown cases, the error decreases with network size.

functions to train both the surface normal field and the boundary normal field. We chose *Siren* because of its ability to approximate high-frequency functions (Fig. 11 in the appendix).

Our surface normal network is $\mathbf{n}_{\hat{\theta}} : \mathbb{R}^d \to \mathbb{R}^d$, mapping the Euclidean coordinates $\mathbf{x} \in \Omega$ to the normal $\mathbf{n}$ at $\mathbf{x}$ (even though the coordinate network takes points in $\mathbb{R}^3$ as input, we only ever evaluate it on Ω). The last layer of the network normalizes the output to ensure unit normals. Therefore, while training, we focus only on the direction of the normals at each point on Ω. We define the loss function

$$\mathcal{L}_{\text{normals}} = \sum_{i=1}^{N} (1 - (\mathbf{n}_{\hat{\theta}_i} \cdot \mathbf{n}_{\text{GT}_i}))^2 \ , \tag{13}$$

where N is the number of uniformly sampled points on the surface Ω. Eq. (13) minimizes the angular deviation between the predicted and ground-truth surface normals, rather than directly penalizing their vector-valued discrepancy using an ℓ_2 loss. An ℓ_2 objective is sensitive to differences in both direction and magnitude, whereas our objective of interest depends solely on the directional alignment of the normals. By optimizing the angular error, we therefore relax the learning objective, allowing the network to recover the correct normal direction without imposing an explicit unit-norm constraint on its predictions. The predicted normals are subsequently normalized to unit length before further use. We empirically validate the effectiveness of this formulation and its advantages over direct ℓ_2 loss in Section B.3. In Section B.1 and Fig. 13 in the appendix, we illustrate how the PDE approximation error changes as the size of the normal network increases. The error initially decreases and then plateaus, indicating that once the network is sufficiently expressive to approximate the normal field accurately, further increase in its size no longer impact the error. Empirically, we observe that a network with width 64 and depth 5 is adequate for learning the normal field. We train the network for $\approx 1e7$ iterations with a learning rate of $1e^{-4}$. The normal network training converges quickly for simple smooth domains but takes more time for complex geometries.

Our boundary normal network $\nu_{\tilde{\theta}} : \mathbb{R}^d \to \mathbb{R}^d$ takes the Cartesian coordinates $\mathbf{x} \in \partial\Omega_N$ and outputs the boundary normal ν at $\mathbf{x}$ (even though the coordinate network takes points in $\mathbb{R}^d$ as input, we only ever evaluate it on $\partial\Omega_N$). The training hyperparameters for $\nu_{\tilde{\theta}}$ are the same as for $\mathbf{n}_{\hat{\theta}}$. We use the same loss function equation 13 where ν_{GT} is computed using the cross product between the boundary edge and the surface normal.

PDE Network u_θ. Once we have the surface normals (and boundary normals) we compute the Laplace-Beltrami operator using equation 6. Note that since the normals are computed using differentiable neural networks $\mathbf{n}_{\tilde{\theta}}$, $\nu_{\tilde{\theta}}$, we can use *autodiff* to train for θ in u_θ. We train our network u_θ to solve for the Poisson and the Helmholtz equation using equation 7 and equation 8 respectively. The boundary loss terms equation 10 and equation 11 are added as soft constraints. Fig. 2 shows an overview of our method.

For all experiments, we choose a starting learning rate of $1e^{-3}$ with the *ReduceLROnPlateau* scheduler. We uniformly sample points on the domain and train until u_θ converges and saturates completely.

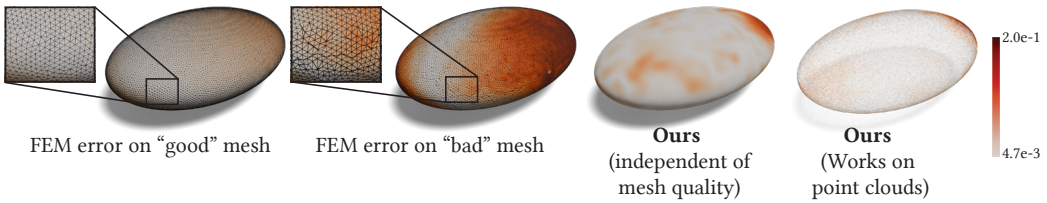


Figure 5: While the FEM error is dependent on the mesh quality – error increases with mesh degradation, our method is independent of mesh quality. Since our method only requires sampling points, it can even work directly on a point cloud with no edge connectivity.

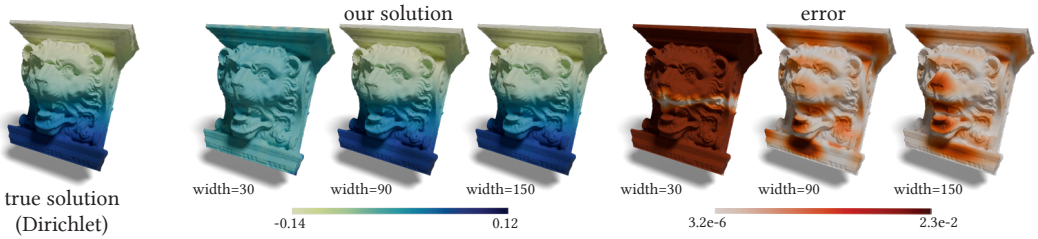


Figure 6: Our method can effectively solve PDEs even on mesh with intricate details.

Empirically, we noticed that the training usually converges around $50k$ iterations (Section C.1 and Fig. 15 in the appendix). We train all the networks for $150k$ iterations to ensure the network is completely saturated.

To study convergence, we train our PDE network u_θ with depth 3 and vary the width of the network from 30 to 150. In Section B.1 and Fig. 12 in the appendix, we show that the choice of varying depth versus width doesn't change the final error. Therefore, without loss of generality, we choose to vary the width in all our experiments. For our convergence study, we consider the degrees of freedom to be the total number of tunable weights+biases in the network ($\#W$). Please see Section D in the appendix for further details. In Section G of the appendix, we additionally give a simple argument, in a simplified setting, for why the error should decrease as the network width grows.

4 Results

4.1 Solving PDEs

We solve the Poisson equation on a wide variety of domains: flat surfaces $\subseteq \mathbb{R}^2$, where our method simplifies to a standard PINN (Fig. 9), curved surfaces with boundary where exact normals and boundary normals are known (Fig. 3(a, b)), and arbitrary manifold surfaces given as triangle meshes, where we train a neural network to have a smooth normal field that approximates the piecewise constant mesh normals (Fig. 3(e)). Beyond the Poisson equation, our method also solves the Helmholtz (Fig. 3(c, d, f)) and heat equations (Fig. 1(a, b)). In Fig. 6, we demonstrate our method's ability to solve the Helmholtz equation on a challenging intricate curved mesh with fine geometric detail. In Fig. 4, we show that our method exhibits convergence behavior for different PDEs (Poisson and Helmholtz), surfaces, boundary conditions and test functions. The details of our experimental setup are provided in Section A in the appendix. Our released code and details about our full convergence study using different test functions on different geometries and boundary conditions is presented in Section F in the appendix.

4.2 Comparison with Baselines

Since our PINN only requires sampling of points for training, unlike FEM, it is not sensitive to mesh quality and can work directly on point clouds (Fig. 5). Classical PINNs such as Kharazmi et al. (2021); Yu et al. (2018) cannot directly solve surface PDEs, so we compare FEM, Kairanda et al. (2026); Kharazmi et al. (2021); Yu et al. (2018), and our method on $\mathbb{R}^2$. Our approach exhibits convergence behavior (Fig. 9), while Kharazmi et al. (2021); Yu et al. (2018) do not converge under the network refinement strategy used here. We attribute this difference partly to the use of Siren,

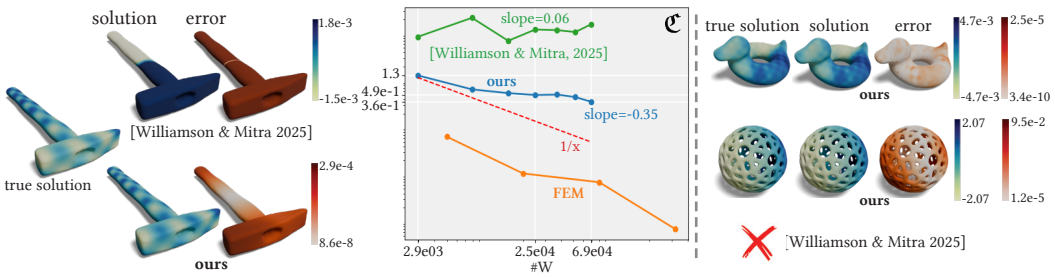


Figure 7: **Comparison with Williamson & Mitra (2025)**. By directly training on surface normals, our method achieves a lower PDE solution error. Moreover, our approach applies to arbitrary topologies, whereas Williamson & Mitra (2025) is limited to surfaces homeomorphic to a sphere.

which better captures the high-frequency solution u compared to the *tanh* of Kharazmi et al. (2021) and $\max(0, x^2)$ of Yu et al. (2018) (Fig. 11 in the appendix). Our method's convergence also stems from the use of the PDE loss (Eq. (7), Eq. (8)) instead of the variational loss used by Kharazmi et al. (2021); Yu et al. (2018). More details can be found in Section E in the appendix. Although Kairanda et al. (2026) demonstrates some convergence with increasing resolution, improvements eventually saturate due to interpolation assumptions. More details about our convergence study can be found in the appendix in Section D.

On surfaces $\Omega \subset \mathbb{R}^3$, we compare against a parameterization-based approach similar to Williamson & Mitra (2025) (Fig. 7). We speculate that their method resulting in a higher error stems from challenges in learning parameterizations and differentiation errors, both avoided by our method. Note that we support arbitrary manifold topologies, unlike Williamson & Mitra (2025). We also compare to Sugimoto et al. (2024) and Hu et al. (2024) (Fig. 10). Sugimoto et al. (2024) is a Monte Carlo method limited to Dirichlet problems with simple source terms, while our approach supports both Dirichlet and Neumann boundaries. Hu et al. (2024) is a neural based method similar to ours but they require an explicit level-set formulation of the underlying surface to compute the normal field which is generally not known. Our method doesn't have such a limitation.

4.3 Geometry Processing Applications

Since our method works reliably on a wide variety of curved surfaces, it can be used directly for a range of geometry processing applications where FEM would traditionally be used.

Minimal Surfaces. A surface $\Omega \subset \mathbb{R}^3$ is minimal if its mean curvature is zero everywhere. The mean curvature relates to the Laplace–Beltrami operator via $\Delta_\Omega = -2H\mathbf{n}$, where H is the mean curvature and $\mathbf{n}$ the surface normal. To compute a minimal surface from a cylinder, we solve three Laplace equations: $\Delta_\Omega u = 0$ with Dirichlet boundary conditions for each coordinate (x, y, z). The resulting solutions define the new vertex positions. This recovers the cylinder's minimal surface, the catenoid (Fig. 8(a)).

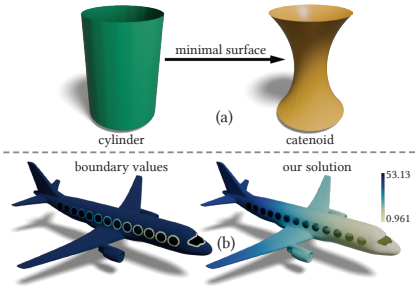


Figure 8: (a) Result of the minimal surface computed for the cylinder. (b) Result of harmonic interpolation in the interior of the plane given Dirichlet boundary values on the windows of the plane.

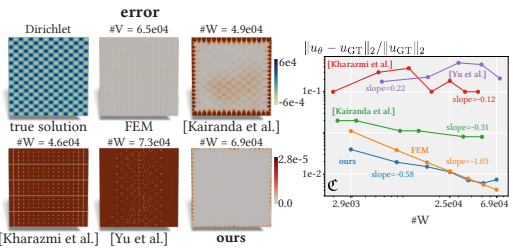
Harmonic Interpolation. In Fig. 8(b), we show that PINNsur can be used to compute the harmonic interpolation of a surface given the Dirichlet boundary condition by solving the Laplace equation

$$\Delta_\Omega u = 0, \quad u(\mathbf{x}) = g(\mathbf{x}), \mathbf{x} \in \partial\Omega_D \ . \tag{14}$$

Here, our method's ability to handle all domain topologies particularly shines, since interpolation examples can have many different disconnected boundaries.

Heat Equation. Laplacians can also be used to solve the heat equation. The implicit Euler finite difference scheme in time for the heat equation $\frac{\partial u}{\partial t} = \Delta_\Omega u$, results in a screened Poisson equation

$$\Delta_\Omega u^{t+1} - h^{-1} u^{t+1} = -h^{-1} u^t \ , \tag{15}$$



Method	Degrees of Freedom		
	2.91e3	2.49e4	5.16e4
FEM	1.11e−1	1.28e−2	**5.55e−3**
(Kairanda et al., 2026)	1.98e−1	9.67e−2	8.06e−2
(Kharazmi et al., 2021)	1.00e0	1.83e0	9.99e−1
(Yu et al., 2018)	1.76e0	5.04e0	4.57e0
ours	**3.97e−2**	**1.13e−2**	6.04e−3

Figure 9: **Poisson solve on a Euclidean domain**. A comparison with FEM, Kairanda et al. (2026), Kharazmi et al. (2021), Yu et al. (2018) for similar degrees of freedom shows that our method attains error similar to FEM and exhibits convergent behavior.

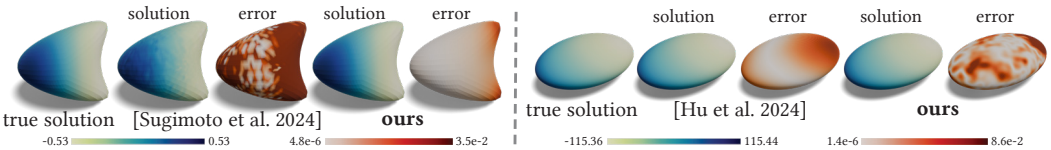


Figure 10: Comparison of our method with (Sugimoto et al., 2024) and (Hu et al., 2024). We show that compared to (Sugimoto et al., 2024), our method achieves a lower error when solving a screened Poisson equation on the Dziuk surface. Compared to (Hu et al., 2024), our method results in a similar error but without the level-set formulation of the surface explicitly known.

where h is the time-step size. PINNsur solves the heat equation on intricate curved surfaces by training for each time step. One-step solution of the heat equation given initial heat sources is shown in Fig. 1(b). We can also compute heat-based geodesic distances on a surface using the method of Crane et al. (2017) as shown in Fig. 1(b).

Mean Curvature Flow. Using the heat equation, we can compute an analog to mean curvature flow (Fig. 1(a)) by solving equation 15 three times for $u^t = x, y, z$ where x, y, z are the mesh vertex coordinates. The solution u^{t+1} forms the new vertex coordinates.

5 Limitations

While we show the effectiveness of our method on a range of surfaces and boundary conditions, our method fails if the surface has a discontinuous normal field. For example, as shown in Section B.2 and Fig. 14 in the appendix, the boundary normals are discontinuous near the corners of the heightfield, and hence the model fails to solve a Poisson equation with Neumann boundary. Our method also fails if the range of the function u is too large. Note that, like other PINNs our method does not always beat FEM in absolute error and runtime (due to training), but it does not require high-quality meshes like FEM and can be used to solve surface PDEs directly on point clouds (Fig. 5).

6 Conclusion & Discussion

We introduce PINNsur, a PINN for solving second-order PDEs such as the Poisson and Helmholtz equations on curved surfaces of arbitrary topology. Prior PINNs either cannot handle surfaces or are limited to genus-0. We also empirically study the convergence of PINNsur and other popular PINNs, finding that PINNsur's error generally decreases as $\#W$ increases (see section 5 for exceptions).

As a first step towards a theoretical analysis, Section G in the appendix shows, for closed surfaces with exact normals, that the solution error is bounded by the loss and that the smallest achievable loss decreases monotonically to zero as the network width grows. This argument provides no rate and does not cover soft boundary constraints, learned normals, or the optimization gap. To obtain rates, we plan to combine expressiveness results for neural networks (Lu et al., 2017) with existing PINN convergence analyses (Section 2), yielding error bounds that depend on the width and depth of the PDE network, and to use scaling laws for PINNs (Lai et al., 2026) to study the effect of other training parameters. Finally, while our current convergence analysis primarily focuses on PDE network width, with preliminary analysis on other hyperparameters such as PDE network depth and normal network, we are also excited to explore more sophisticated refinement strategies in future work.

REFERENCES

Noam Aigerman, Kunal Gupta, Vladimir G. Kim, Siddhartha Chaudhuri, Jun Saito, and Thibault Groueix. Neural jacobian fields: learning intrinsic mappings of arbitrary meshes. *ACM Transactions on Graphics*, 2022.

Jithu J Athalathil, Bhargav Vaidya, Sayan Kundu, Vishal Upendran, and Mark C. M. Cheung. Surface flux transport modeling using physics-informed neural networks. *The Astrophysical Journal*, 975(2):258, November 2024. ISSN 1538-4357. doi: 10.3847/1538-4357/ad7d91. URL http://dx.doi.org/10.3847/1538-4357/ad7d91.

Jithu J. Athalathil, Mohammed H. Talafha, and Bhargav Vaidya. Investigating nonlinear quenching effects on polar field buildup in the sun using physics-informed neural networks. *The Astrophysical Journal*, 1000(1):79, March 2026. ISSN 1538-4357. doi: 10.3847/1538-4357/ae469c. URL http://dx.doi.org/10.3847/1538-4357/ae469c.

Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. *ACM Trans. Graph.*, 26(3):72–es, 2007.

Jan Bednarik, Shaifali Parashar, Erhan Gundogdu, Mathieu Salzmann, and Pascal Fua. Shape reconstruction by learning differentiable surface representations. In *Computer Vision and Pattern Recognition (CVPR)*, 2020.

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon Mesh Processing*. Taylor & Francis, 2010.

Susanne C. Brenner and L. Ridgway Scott. *The Mathematical Theory of Finite Element Methods*. Springer, 2008.

Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (pinns) for fluid mechanics: a review. *Acta Mechanica Sinica*, 2021.

Yue Chang, Otman Benchekroun, Maurizio M Chiaramonte, Peter Yichen Chen, and Eitan Grinspun. Shape space spectra. *ACM Transactions on Graphics (TOG)*, 2025.

Honglin Chen, Rundi Wu, Eitan Grinspun, Changxi Zheng, and Peter Yichen Chen. Implicit neural spatial representations for time-dependent pdes. In *International Conference on Machine Learning (ICML)*, 2023.

Aditya Chetan, Guandao Yang, Zichen Wang, Steve Marschner, and Bharath Hariharan. Accurate differential operators for hybrid neural fields. In *Computer Vision and Pattern Recognition (CVPR)*, 2025.

Keenan Crane, Clarisse Weischedel, and Max Wardetzky. The heat method for distance computation. *Commun. ACM*, 60(11):90–99, October 2017. ISSN 0001-0782. doi: 10.1145/3131280. URL https://doi.org/10.1145/3131280.

Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 317–324, 1999.

Gerhard Dziuk. *Finite Elements for the Beltrami operator on arbitrary surfaces*, pp. 142–155. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988.

Venkataram Edavamadathil Sivaram, Tzu-Mao Li, and Ravi Ramamoorthi. Neural geometry fields for meshes. In *ACM SIGGRAPH 2024 Conference Papers*, 2024.

Lawrence Evans. *Partial Differential Equations*. American Mathematical Society, March 2010. ISBN 9781470411442. doi: 10.1090/gsm/019. URL http://dx.doi.org/10.1090/gsm/019.

Guy Fargion and Ofir Weber. Learning Metric Fields for Fast Low-Distortion Mesh Parameterizations. *Computer Graphics Forum*, 2025.

Zachary Ferguson, Pranav Jain, Denis Zorin, Teseo Schneider, and Daniele Panozzo. High-order incremental potential contact for elastodynamic simulation on curved meshes. In *ACM SIGGRAPH 2023 Conference Proceedings*, 2023.

Alexander Gao, Maurice Chu, Mubbasir Kapadia, Ming C. Lin, and Hsueh-Ti Derek Liu. An intrinsic vector heat network. In *International Conference on Machine Learning (ICML)*, 2024.

Zhongkai Hao, Songming Liu, Yichi Zhang, Chengyang Ying, Yao Feng, Hang Su, and Jun Zhu. Physics-informed machine learning: A survey on problems, methods and applications. *arXiv preprint arXiv:2211.08064*, 2022.

Anil Hirani. *Discrete Exterior Calculus*. PhD thesis, California Institute of Technology, 2003.

Wei-Fan Hu, Yi-Jun Shih, Te-Sheng Lin, and Ming-Chih Lai. A shallow physics-informed neural network for solving partial differential equations on static and evolving surfaces. *Computer Methods in Applied Mechanics and Engineering*, 418:116486, January 2024. ISSN 0045-7825. doi: 10.1016/j.cma.2023.116486. URL http://dx.doi.org/10.1016/j.cma.2023.116486.

John H. Hubbard and Barbara Burke Hubbard. *Vector Calculus, Linear Algebra, and Differential Forms: A Unified Approach (5th edition)*. Matrix Editions, September 2015. URL https://hal.science/hal-01297648.

Pranav Jain, Ziyin Qu, Peter Yichen Chen, and Oded Stein. Neural monte carlo fluid simulation. In *ACM SIGGRAPH 2024 Conference Papers*, 2024.

Navami Kairanda, Marc Habermann, Christian Theobalt, and Vladislav Golyanik. Neuralclothsim: Neural deformation fields meet the thin shell theory. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Navami Kairanda, Shanthika Naik, Marc Habermann, Avinash Sharma, Christian Theobalt, and Vladislav Golyanik. ∂^∞-grid: A neural differential equation solver with differentiable feature grids. In *International Conference on Learning Representations (ICLR)*, 2026.

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 2021.

Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics*, 32(3):1–13, June 2013. ISSN 1557-7368. doi: 10.1145/2487228.2487237. URL http://dx.doi.org/10.1145/2487228.2487237.

Ehsan Kharazmi, Zhongqiang Zhang, and George E.M. Karniadakis. hp-vpinns: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 374:113547, February 2021. ISSN 0045-7825. doi: 10.1016/j.cma.2020.113547. URL http://dx.doi.org/10.1016/j.cma.2020.113547.

Gyeonghoon Ko and Juho Lee. Riemannian diffusion models on general manifolds via physics-informed neural networks. *ArXiv*, abs/2605.31106, 2026. URL https://api.semanticscholar.org/CorpusID:288814786.

Jasen Lai, Sifan Wang, Chunmei Wang, and Yiping Lu. Unveiling the scaling law of PINNs under non-euclidean geometry, 2026. URL https://openreview.net/forum?id=7NS4qtPM2Q.

Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: a view from the width. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6232–6240, 2017.

Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. A level set theory for neural implicit evolution under explicit flows. In *European Conference on Computer Vision (ECCV)*, 2022.

Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *International Workshop on Visualization and Mathematics*, 2002. URL https://api.semanticscholar.org/CorpusID:11850545.

Luca Morreale, Noam Aigerman, Vladimir G Kim, and Niloy J Mitra. Neural surface maps. In *Computer Vision and Pattern Recognition (CVPR)*, 2021.

Luca Morreale, Noam Aigerman, Paul Guerrero, Vladimir G Kim, and Niloy J Mitra. Neural convolutional surfaces. In *Computer Vision and Pattern Recognition (CVPR)*, 2022.

Hong Chul Nam, Julius Berner, and Anima Anandkumar. Solving poisson equations using neural walk-on-spheres. *International Conference on Machine Learning (ICML)*, 2024.

Tiago Novello, Guilherme Schardong, Luiz Schirmer, Vinícius Da Silva, Hélio Lopes, and Luiz Velho. Exploring differential geometry in neural implicits. *Computers & Graphics*, 2022.

Tiago Novello, Vinícius da Silva, Guilherme Schardong, Luiz Schirmer, Hélio Lopes, and Luiz Velho. Neural implicit surface evolution. In *International Conference on Computer Vision (ICCV)*, 2023.

Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experiment. Math*, 2:15–36, 1993.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 2019.

Chengping Rao, Hao Sun, and Yang Liu. Physics-informed deep learning for computational elastodynamics without labeled data. *Journal of Engineering Mechanics*, 2021.

Yeonjong Shin, Jerome Darbon, and George Em Karniadakis. On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes. *arXiv preprint arXiv:2004.01806*, 2020.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Jonathan D. Smith, Kamyar Azizzadenesheli, and Zachary E. Ross. Eikonet: Solving the eikonal equation with deep neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 2021.

O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pp. 175–184, 2004.

Ryusuke Sugimoto, Nathan King, Toshiya Hachisuka, and Christopher Batty. Projected walk on spheres: A monte carlo closest point method for surface pdes. In *SIGGRAPH Asia 2024 Conference Papers*, 2024.

Zhuochao Tang and Zhuojia Fu. Physics-informed neural networks for elliptic partial differential equations on 3d manifolds. *arXiv preprint arXiv:2103.02811*, 2021.

Max Wardetzky. *Discrete Differential Operators on Polyhedral Surfaces – Convergence and Approximation*. PhD thesis, Freie Universität Berlin, 2007.

Lilian Welschinger, Yilin Liu, Zican Wang, and Niloy Mitra. Learning to solve pdes on neural shape representations. *arXiv preprint arXiv:2512.21311*, 2025.

Romy Williamson and Niloy J. Mitra. Neural geometry processing via spherical neural surfaces. *Compututer Graphics Forum*, 2025.

Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. Geometry processing with neural fields. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Bing Yu et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 2018.

Jonas Zehnder, Yue Li, Stelian Coros, and Bernhard Thomaszewski. Ntopo: Mesh-free topology optimization using implicit neural representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

PHYSICS-INFORMED NEURAL NETWORKS FOR PDES ON CURVED SURFACES —APPENDICES—

A Experimental Setup **13**

B Further Experiments **14**

B.1 Further Ablations 14

B.2 Limitation Examples 16

B.3 Comparing Normals Using Eq. (13) Compared to ℓ_2 Loss 16

C Computational Timings **16**

C.1 Training Convergence Plots 18

D Convergence Study **18**

D.1 Convergence Experiments 18

D.2 Ablation Study 19

E Additional Analysis of the Effect of Network Architecture and Training Objective on Convergence **19**

F Source Code Access and Extended Convergence Experiments **21**

G Why the Error Decreases with Network Refinement **22**

A EXPERIMENTAL SETUP

We set up experiments by first defining a pair of functions u and f on the given surface Ω that ensure homogeneous boundary conditions and the compatibility condition equation 12. We define analytic u, f from a family of polynomial and trigonometric functions for surfaces whose normal field is known exactly. For surfaces with boundary and no analytic form of the normal field, we use the eigenfunctions of the Laplace-Beltrami operator ϕ as the function u. This gives

$$\Delta_\Omega \phi_i = \lambda_i \phi_i = f, \tag{16}$$

where ϕ_i and λ_i are the ith eigenfunction and eigenvalue.
Similarly, for the Helmholtz equation

$$\Delta_\Omega \phi_i + k^2 \phi_i = (\lambda_i + k^2)\phi_i = f. \tag{17}$$

For surface meshes with no boundary and no analytic normal field, we define the source terms f from a family of polynomial and trigonometric functions and compute u using the cotangent Laplacian on a fine mesh. To ensure compatibility, we compute $\int_\Omega f dA$ using the mass matrix of the triangle mesh and subtract it from f.

After defining the pairs u and f, we set up our network with different widths and train each of them until complete saturation. Once every network is converged, we compute the relative ℓ_2 error and plot it on a log-log plot to judge convergence as discussed in Section D. Note that for surfaces where the true analytical solution is not known, we compute u_{GT} by solving

$$-L u_{\text{GT}} = M f \tag{18}$$

for the Poisson equation, where L and M are the cotangent Laplacian and the mass matrix of a fine triangular mesh, respectively. Similarly, for the Helmholtz equation, we solve

$$(-L + k^2 I)u_{\text{GT}} = Mf. \tag{19}$$

Given an analytical solution (or FEM solution with cotangent Laplacian) on a surface Ω, we study the convergence patterns by training a PDE network u_θ of varying widths keeping the depth constant, and we plot the log of relative ℓ_2 error with respect to the network parameters $\#W$ for each network after it's fully trained (Fig. 4).

B Further Experiments

B.1 Further Ablations

In Fig. 11, we show that the use of the *Siren* network with *sine* activation function results in a much lower error compared to using *tanh* or *sigmoid* activation functions. This experiment justifies our use of *Siren* network both for the normal and the PDE network.

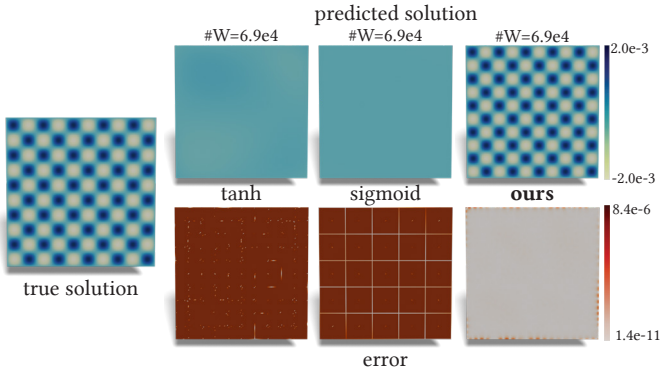


Figure 11: An ablation study where we compare the *Siren* network, which uses $sine$ activation function with *Tanh* and *Sigmoid* activation functions on a 2D domain. The figure shows that *Tanh* and *Sigmoid* fail to solve for a high-frequency function.

Since we use the number of tunable weights $\#W$ as our degrees of freedom, there are two ways to increase $\#W$ for an MLP – by increasing the depth or by increasing the width of the MLP. In Fig. 12, we show that both choices result in the same error for equal number of tunable weights $\#W$. In all our experiments we choose to increase the width of the network to increase $\#W$.

We also repeated the experiment in Fig. 12 by changing both the width and depth at the same time and the results are reported in the table below:

Width, Depth	Relative Error
30, 3	0.7950863058568903
50, 4	0.6608337472213991
70, 5	0.5318795453791865
90, 6	0.1850016993524315

The results demonstrate that the accuracy consistently improves as the total number of trainable parameters increases, regardless of how those parameters are distributed across the network's width or depth. This highlights that model capacity $(\#W)$, rather than the particular choice of width or depth, is the primary factor governing performance.

We further repeat the experiment in Fig. 12 using larger networks with up to $270k$ trainable parameters and the errors are as follows

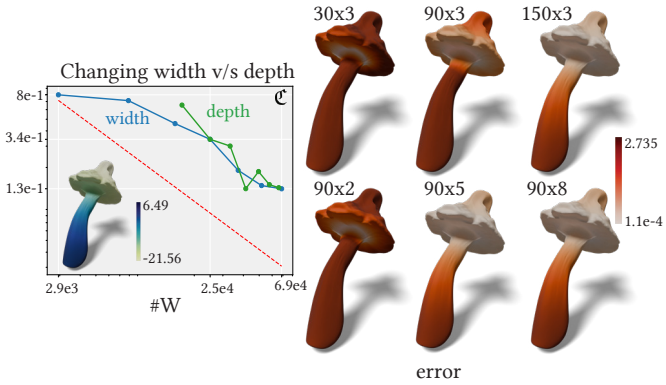


Figure 12: Result of an ablation study where we compare the final error first by changing the width, keeping the depth constant, and vice-versa. The figure shows that the final error in the solution to the PDE doesn't change when the network width is changed, compared to when the depth is changed.

$\#W$	Relative Error
2941 (30x3)	0.79508631
7901 (50x3)	0.71243081
15261 (70x3)	0.46028739
25021 (90x3)	0.34145211
37181 (110x3)	0.18947342
51741 (130x3)	0.14177541
68701 (150x3)	0.13336032
121601 (200x3)	0.10994321
189501 (250x3)	0.06646518
272401 (300x3)	0.06387724

Note that the error decreases monotonically with weights with slope ≈ -0.63 showing that the error doesn't saturate.

In Fig. 13, we show how the final error behaves as the normal network size is increased. We observe that the error first decreases and then stays constant showing that once the normal network is large enough to accurately learn the surface normals, the size of the normal network doesn't influence the final PDE error.

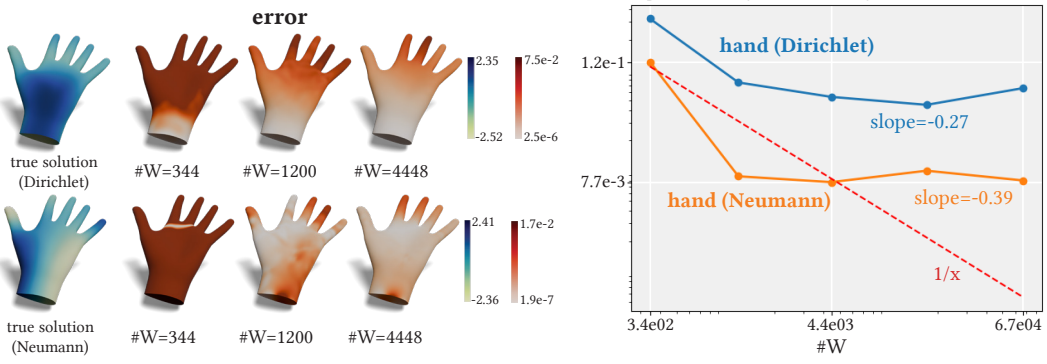


Figure 13: Figure showing how the error varies when the tunable parameters of the normal network are increased keeping the PDE network parameters constant.

B.2 Limitation Examples

Fig. 14 illustrates the limitations of our approach where our network fails to solve for the given PDE. Enforcing Neumann boundary conditions becomes challenging in regions where the normal is discontinuous (e.g., at the corners of the height field). Additionally, the method breaks down when the solution u exhibits large variance.

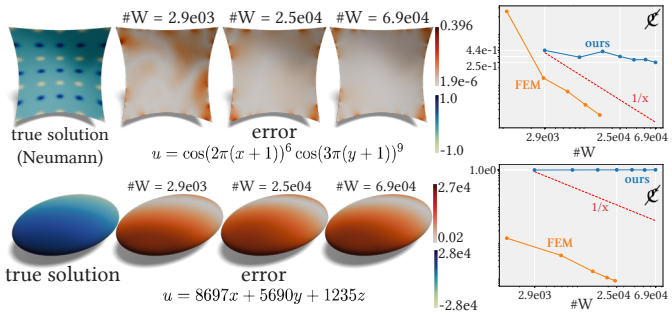


Figure 14: Figure showing the limitations of our work. Enforcing Neumann boundary is difficult at regions where the normal is discontinuous (for ex., at the corners of the heightfield). The method also fails if the solution u has a huge variance.

B.3 Comparing Normals Using Eq. (13) Compared to ℓ_2 Loss

We ran the experiment for training the normals on the model as in Fig. 6 with our loss (Eq. (13)) and ℓ_2 error between the predicted and the true normal and report the losses at every 20000 iterations for the first 100k iterations:

Iterations	Errors using Eq. (13)	Errors using ℓ_2 loss
0	8296.26171875	14218.56640625
20k	37.8331718444	2074.925048828
40k	7.384617805480	1456.8861083984
60k	1.978981494903	1095.2497558593
80k	0.671690285205	901.52264404296
100k	0.317917495965	709.40600585937

This experiment shows that the loss goes down faster using our proposed angular loss, and thus justifies our choice.

C Computational Timings

We ran all the experiments on an Intel i7 5.2GHz with a NVIDIA RTX 4080 GPU. We report the timings for each example shown in Fig. 3 (Figure showing our solution and the error as the network width sizes are increased for Poisson and Helmholtz equations with different boundary conditions) below. Note that we train each model for 150000 iterations to ensure saturation but most models converge way before 150000 iterations. Therefore, we report timing until convergence and also for full 150000 iterations.

The computational time for solving the Poisson equation with Dirichlet boundary on the heightfield example (Fig. 3a) is as follows:

Width, Depth	Runtime (150000 iters)	Iterations to Convergence	Runtime to Convergence
30, 3	26m 16s	51039	8m 56s
90, 3	29m 22s	69488	13m 42s
150, 3	37m 58s	33319	8m 26s

The computational time for solving the Helmholtz equation with Neumann boundary on the heightfield example (Fig. 3b) is as follows:

Width, Depth	Runtime (150000 iters)	Iterations to Convergence	Runtime to Convergence
30, 3	29m 49s	61109	11m 48s
90, 3	35m 20s	54123	12m 54s
150, 3	46m 24s	41125	12m 40s

The computational time for solving the Helmholtz equation with Dirichlet boundary on the hand example (Fig. 3c) is as follows:

Width, Depth	Runtime (150000 iters)	Iterations to Convergence	Runtime to Convergence
30, 3	19m 47s	38630	5m 12s
90, 3	24m 50s	33408	5m 46s
150, 3	32m 40s	34051	7m 32s

The computational time for solving the Helmholtz equation with Neumann boundary on the hand example (Fig. 3d) are as follows:

Width, Depth	Runtime (150000 iters)	Iterations to Convergence	Runtime to Convergence
30, 3	27m	35826	6m 46s
90, 3	34m	35995	8m 21s
150, 3	41m 46s	26786	7m 38s

The computational time for solving the Poisson equation on the bunny example (Fig. 3e) are as follows:

Width, Depth	Runtime (150000 iters)	Iterations to Convergence	Runtime to Convergence
30, 3	17m 23s	46481	5m 29s
90, 3	22m 47s	41163	6m 11s
150, 3	30m	29613	5m 53s

The computational time for solving the Helmholtz equation on the spot example (Fig. 3f) are as follows:

Width, Depth	Runtime (150000 iters)	Iterations to Convergence	Runtime to Convergence
30, 3	18m 8s	48626	6m
90, 3	23m 10s	29557	4m 39s
150, 3	29m	32337	6m 31s

For normal and boundary normal networks, we train a fixed size network (width 64 and depth 5) for $1e7$ iterations for all examples. The normals are computed either analytically (if known) or using the mesh normals. The computational time for training the normal network is about $2h40m$ (1050 iterations per second).

C.1 Training Convergence Plots

Fig. 15 shows the training loss for different values of $\#W$ on the "bunny" model shown in Fig. 3e. The network typically converges around $50k$ iterations. However, we train for $150k$ iterations to ensure full saturation.

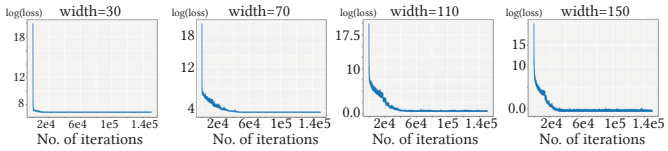


Figure 15: Figure showing the training loss for different $\#W$. The network usually converges around 50k iterations, but we train for 150k iterations to ensure complete saturation of the network.

D Convergence Study

To quantify the accuracy of a PDE solver, we need a notion of convergence. Convergence describes whether the estimate approaches the true solution with respect to a chosen norm. Here we use the relative ℓ_2 norm and define error as

$$e := \|u_\theta - u_{\text{GT}}\|_2 / \|u_{\text{GT}}\|_2 \ , \tag{20}$$

where u_θ and u_{GT} are our optimized solution and the ground-truth, evaluated respectively on $\mathbf{x} \in \Omega$.

In classical discretization-based solvers such as finite-element schemes, convergence is often expressed in relation to refining the number of elements ($\sim$ degrees of freedom) n of the discretization (or a quantity similar to the element size) (Brenner & Scott, 2008). Convergence is observed if the error decreases as more elements are added to the discretization. In our method (and more broadly in neural solvers that model continuous fields), there is no discretization in the classical sense (and therefore no direct refinement analog). Instead, the weights of the neural field induce an implicit discretization by limiting the model's expressive capacity to represent the continuous solution. We therefore treat the total number of tunable neural network weights ($\#W$) as degrees of freedom and study convergence under the expectation that the error $e \to 0$ as $\#W \to \infty$. We compute $\#W$ as the number of parameters of the *Siren* network that models the PDE solution u_θ. Since *Siren* is a coordinate-based MLP, $\#W$ is determined by the depth and width of the MLP. We exclude the normal network $\mathbf{n}_{\hat{\theta}}$ from this count, since it is pretrained and unaffected by the PDE. For u_{GT}, we use analytical solutions when available; otherwise, we use a high-resolution FEM solution as GT (cotangent Laplacian (Meyer et al., 2002) on the surface Ω).

For our convergence study, we examine whether the error e in equation 20 decreases as the number of tunable weights (degrees of freedom) increases. Accordingly, we interpret a consistent decay in the error with increasing model size as evidence of convergence.

To quantify whether our network shows convergent behavior, we fit a line via least-squares linear regression of $\log e$ on $\log(\#W)$ and obtain the slope m and Pearson correlation coefficient r. An experiment is deemed to exhibit convergence (indicated by $\mathfrak{C}$ in figures) if $m < -0.3$ and $r < -0.5$, indicating a sufficiently steep decay and a reasonably linear trend. We emphasize that the chosen convergence heuristic is purely empirical, chosen to promote a stable and monotonic decrease in approximation error as the degrees of freedom increase, while mitigating oscillations arising from noise.

In Section G, we additionally give a simple argument, in a simplified setting, for why the error should decrease as the network width grows.

D.1 Convergence Experiments

Using our convergence criterion established in Section D, we evaluate and observe the convergence of our method for both Poisson and Helmholtz equations on a wide variety of surfaces with different

boundary conditions using different source terms (see Fig. 4). We compare convergence rates with Kharazmi et al. (2021); Yu et al. (2018); Kairanda et al. (2026) and FEM on a flat 2D domain (Fig. 9). On curved surfaces, we compare against FEM and a parameterized PINN (Williamson & Mitra, 2025) (see Section 4.2 for more details). The runtimes for each convergence experiment in Fig. 3 can be found in Section C. In Section F, we present our code and the setting of our numerical experiments to test convergence for different test functions, geometries and boundary conditions.

D.2 Ablation Study

To justify our parameter choices, we perform ablations. In all our experiments, we vary the network width while keeping the depth constant. Fig. 12 shows that the choice of altering depth instead of width gives similar results. In Fig. 11, we show that the *Siren* network is much better at learning high-frequency functions compared to using *sigmoid* or *tanh* activation functions. In Fig. 13, we research the dependence of the error on the normal network size. The error initially decreases and then remains constant, indicating that once the normal network is expressive enough, only the main network weights matter.

We provide additional ablations in Section E, demonstrating that the observed convergence is not specific to SIREN and depends critically on the choice of training objective.

E Additional Analysis of the Effect of Network Architecture and Training Objective on Convergence

In this section, we provide additional experiments to disentangle the effects of the network architecture, activation function, and training objective on convergence. In particular, we investigate whether the convergence behavior observed in our method can be attributed solely to the use of SIREN, or whether the choice of PDE-based training objective is also essential.

Effect of SIREN and the PDE residual objective. We first repeat the experiment presented in Fig. 9 by progressively modifying the method of Yu et al. (2018). On a flat 2D domain, our formulation reduces to a standard physics-informed neural network (PINN). The method of Yu et al. (2018) differs from this formulation primarily in two aspects: (i) its network architecture incorporates residual connections, and (ii) it minimizes a variational formulation of the governing PDE rather than the pointwise PDE residual (Eq. (7), Eq. (8)). To isolate the contribution of these components, we consider two successive modifications to the method of Yu et al. (2018). First, we replace its Tanh-based network with SIREN while retaining the original variational objective. Second, we additionally replace the variational objective with the PDE residual loss defined in Eq. (7). The resulting errors for different network widths are reported in the table below.

Width x Depth	Error (Yu et al. (2018) with Tanh)	Error (Yu et al. (2018) with Siren)	Error (Yu et al. (2018) with Siren and PDE loss)
30 x 3	1.762194473546379	1.0015226509184367	0.08354167491330658
50 x 3	2.263583299512813	1.0044855329636857	0.02592456856874785
70 x 3	5.040743013328354	1.010064576416256	0.02340237775862254
90 x 3	4.576080187619759	1.004759674709261	0.01671808983891341
110 x 3	2.105006810882283	1.0200976521471248	0.012796287510063311
130 x 3	1.1425196017745296	1.0126271150737571	0.006547458165332402
150 x 3	1.2263844731734532	1.0800892567104916	0.005583985290001128

Replacing Tanh with SIREN alone does not lead to convergence: across all network widths, the error remains approximately $\mathcal{O}(1)$. In contrast, when the variational objective is additionally replaced by the PDE residual loss, the approximation error decreases substantially and consistently with increasing network width. These results indicate that the observed convergence cannot be attributed

solely to the spectral properties of the SIREN architecture. Rather, the choice of training objective plays a critical role.

This observation is further supported by applying the same activation-function modification to the method of Kharazmi et al. (2021). Replacing Tanh with SIREN while retaining its variational formulation again fails to produce convergence, with the error remaining close to $\mathcal{O}(1)$ across network widths. Taken together, these experiments suggest that replacing a conventional activation function with SIREN is, by itself, insufficient to recover the convergence behavior observed with our formulation. Instead, convergence arises from the combination of an appropriate function representation and the PDE residual-based training objective.

Width x Depth	Error (Kharazmi et al. (2021) with tanh)	Error (Kharazmi et al. (2021) with Siren)
30 x 3	1.264892218430015	1.0411572828294617
50 x 3	2.97057167671635	1.109605389425515
70 x 3	3.7225744819142275	1.0175606991220487
90 x 3	1.003352973357556	1.00302601384504
110 x 3	1.83340966477261	1.2033078111924245
130 x 3	1.003095818157352	1.0000595795946177
150 x 3	0.9999993575722375	1.0084170883250239

Is convergence specific to SIREN? We next investigate the converse question: whether SIREN is necessary for the convergence behavior of our formulation. As demonstrated in Fig. 11, Tanh and Sigmoid networks have difficulty representing the high-frequency solution considered in our main experiments. We therefore consider a simpler Poisson problem on a flat 2D domain with Dirichlet boundary conditions ($u = (x-1)(x+1)(y-1)(y+1)/(2(x-1))^2$) and compare our method using SIREN and Tanh activations.

Width x Depth	Error (using Siren)	Error (using Tanh)
30 x 3	0.00020466071	0.005700733
50 x 3	8.4626576e-05	0.0010309056
70 x 3	3.1031097e-05	0.00033634875
90 x 3	3.0415771e-05	0.0001473034
110 x 3	1.5698854e-05	0.00011790357
130 x 3	1.2892722e-05	0.00024639102
150 x 3	8.9295645e-06	0.00009552111

We observe that SIREN consistently achieves lower approximation errors. Nevertheless, the Tanh network also exhibits a clear reduction in error as the network width increases. Thus, although the choice of activation function substantially affects approximation accuracy, particularly for high-frequency solutions, the convergence behavior of our formulation is not specific to SIREN.

Together, these experiments distinguish the roles of the representation and the optimization objective. SIREN provides a more suitable function representation for high-frequency solutions, whereas the PDE residual objective is important for obtaining the convergence behavior observed in our experiments. In particular, SIREN alone does not recover convergence for the variational formulations of Yu et al. (2018) and Kharazmi et al. (2021), while our PDE residual-based formulation continues to exhibit convergence when SIREN is replaced by Tanh on a problem that can be adequately represented by a Tanh network. These results therefore suggest that the observed convergence is not an artifact of a particular activation function, but depends critically on the formulation of the learning objective.

Geometry / BC	Exact normals	MLP: exact normals	MLP: mesh normals
Heightfield, Dirichlet	10/10	10/10	8/10
Heightfield, Neumann	6/10	7/10	6/10
Ellipsoid, no boundary	8/12	8/12	9/12

Table 1: Results from the extended convergence experiments included in the released code. The entries report the number of individual test functions exhibiting empirical convergence under network refinement. The complete suite deliberately includes challenging manufactured solutions used to probe the limitations of the method.

F Source Code Access and Extended Convergence Experiments

Our complete code is available at the accompanying repository.[1] The `README.md` contains instructions for setting up the environment, running the experiments, and reproducing the results reported in the paper. In addition to the representative experiments presented in the main paper, the released code contains a larger suite of manufactured-solution experiments that we used throughout development to investigate the regimes in which the proposed neural PDE solver succeeds or fails.

Extended convergence test suite. The experiments were constructed with varying target functions, boundary conditions, surface geometries, and representations of the surface normal field. For the heightfield and ellipsoid, we consider three variants: (i) using the exact surface normals directly, (ii) training an MLP to approximate the exact surface normals, and (iii) training an MLP to approximate normals computed from a triangle mesh. We additionally evaluate the method on arbitrary meshes for which only mesh-derived normals are available.

Table 1 summarizes the results contained in the released code. Each entry reports the number of individual manufactured solutions for which the approximation error decreases under network refinement according to the empirical test used in our analysis, out of the total number of functions considered for that setting.

For comparison, when the Poisson equation is solved directly on its parameter domain, without introducing the curved-surface differential operator, 10/10 Dirichlet and 9/10 Neumann test functions exhibit convergence on the planar domain, while 4/4 tests converge on the sphere. On arbitrary meshes, we observe convergence for 10/10 Dirichlet and 10/10 Neumann experiments on the hand model, 10/10 experiments on the bunny, and 9/10 experiments on Spot.

These results make explicit that PINNsur does not converge for every manufactured solution in our extended test suite. The purpose of the extended suite is not only to demonstrate successful examples, but also to probe the regimes in which the observed convergence behavior deteriorates.

Analysis of failure cases. We further investigated the non-convergent cases by systematically increasing the frequency and dynamic range of the manufactured solutions. We find that the failures in the extended test suite are concentrated in the challenging regimes already identified in the limitations of our method (Section 5): functions with large dynamic range, and, for Neumann problems, geometries for which the required normal field contains or is affected by discontinuities.

For the heightfield (with discontinuous boundary normal field) with Poisson–Neumann boundary conditions, the reduced success rate persists across all three representations of the surface normal field: 6/10 test functions converge when exact normals are supplied directly, 7/10 when an MLP is trained to reproduce the exact normals, and 6/10 when an MLP is trained using mesh-computed normals. Representative non-convergent cases correspond to manufactured solutions with either substantially increased spatial frequency or substantially increased function range. Importantly, these failures are also observed when exact surface normals are provided to the PDE solver. They therefore cannot, in general, be attributed solely to approximation error in the learned surface-normal network.

[1] `https://anonymous.4open.science/r/PINNsur_ICLR2027/README.md`

We observe analogous behavior on the ellipsoid. Of the twelve manufactured solutions in the extended ellipsoid suite, 8/12 exhibit convergence with exact normals, 8/12 with learned exact normals, and 9/12 with learned mesh normals. The difficult cases again occur for sufficiently oscillatory or high-range functions. In these regimes, increasing the number of trainable parameters does not necessarily result in a consistent reduction of the approximation error.

These observations are consistent with the limitations discussed in Section 5 and Section B.2. In particular, our method is not expected to reliably converge when the solution contains frequencies that are difficult for the finite neural representation and optimization procedure to resolve, when the solution has a very large dynamic range, or when the normal field required for imposing the PDE or Neumann boundary condition is discontinuous. Fig. 14 provides explicit examples of these failure modes.

Scope of the empirical convergence results. We emphasize that the convergence results in this work constitute an *empirical convergence study*, rather than a theoretical convergence guarantee for arbitrary PDEs or arbitrary target functions. Our experiments study how the approximation error behaves as the number of neural degrees of freedom is increased while the remainder of the experimental setup is held fixed.

The representative experiments in the main paper demonstrate decreasing approximation error across different PDEs, boundary conditions, and surface geometries. The larger suite released with the code complements these experiments by deliberately probing more difficult manufactured solutions and exposing cases in which this behavior breaks down. In particular, the observed failures for high-frequency and high-range solutions provide additional empirical evidence for the limitations described in Section 5.

G Why the Error Decreases with Network Refinement

In this section, we give a simple argument for why the error e in equation 20 should decrease as the width of the PDE network grows. We consider the simplest setting: a smooth, connected surface Ω without boundary whose normal field $\mathbf{n}$ is known exactly, and the equation $\Delta_\Omega u + k^2 u = f$, which is the Helmholtz equation and, for $k = 0$, the Poisson equation.

As in Section A, let ϕ_i and λ_i be the eigenfunctions and eigenvalues of Δ_Ω, with $0 = \lambda_0 > \lambda_1 \geq \lambda_2 \geq \dots$ and the ϕ_i orthonormal. We assume $k^2 \neq -\lambda_i$ for all i, so that the solution u_{GT} is unique, and set $\gamma := \min_i |\lambda_i + k^2| > 0$. For the Poisson equation, a solution exists when $\int_\Omega f \, dA = 0$ (Eq. (12)), but it is unique only up to a constant. We therefore compare u_θ and u_{GT} after subtracting their means, which does not change the loss below, and set $\gamma = |\lambda_1|$.

In the smooth setting, the loss function is defined as

$$\mathcal{L}(u_\theta) = \int_\Omega \left(\Delta_\Omega u_\theta + k^2 u_\theta - f\right)^2 dA, \tag{21}$$

where $\Delta_\Omega u_\theta$ is computed using equation 6. Let $\mathcal{L}_N^\star$ be the smallest value of $\mathcal{L}$ that a PDE network of width N (at the fixed depth of Section 3.4) can attain.

Proposition 1. *Consider the following statements,*

(a) $\mathcal{L}_{N+1}^\star \leq \mathcal{L}_N^\star$ *for every width* N.

(b) Every network u_θ *satisfies* $e \leq \sqrt{\mathcal{L}(u_\theta)} \,/\, \left(\gamma \, \|u_{GT}\|\right)$.

(c) $\mathcal{L}_N^\star \to 0$ *as* $N \to \infty$.

Suppose training at width N *reaches a loss* $\mathcal{L}(u_{\theta_N}) \leq \mathcal{L}_N^\star + \eta_N$, *where* $\eta_N \geq 0$ *is the gap to the smallest achievable loss, caused by the numerical errors in optimization and by the finite number of sample points. Then*

$$e \;\leq\; \frac{\sqrt{\mathcal{L}_N^\star + \eta_N}}{\gamma \, \|u_{GT}\|}. \tag{22}$$

From the statements above, $\mathcal{L}^\star_N$, decreases monotonically to zero as the width grows. Therefore, the error decreases as long as η_N decreases.

Proof. Proving each statement independently,

(a) A network of width N is a network of width $N+1$ in which the outgoing weights of the additional neurons are zero. Hence the smallest attainable loss cannot increase with N.

(b) Write the error on Ω as $w = u_\theta - u_{\text{GT}} = \sum_i w_i \phi_i$. Since $\Delta_\Omega u_{\text{GT}} + k^2 u_{\text{GT}} = f$, the residual inside $\mathcal{L}(u_\theta)$ is $\Delta_\Omega w + k^2 w = \sum_i (\lambda_i + k^2)\, w_i \phi_i$; here we used that Δ_Ω is symmetric on a closed surface, so the ith coefficient of $\Delta_\Omega w$ is $\int_\Omega w\, \Delta_\Omega \phi_i = \lambda_i w_i$. Since the ϕ_i form an orthonormal basis, Parseval's identity gives

$$\mathcal{L}(u_\theta) = \sum_i (\lambda_i + k^2)^2 w_i^2 \;\geq\; \gamma^2 \sum_i w_i^2 = \gamma^2 \, \|u_\theta - u_{\text{GT}}\|^2.$$

Dividing by $\gamma^2 \|u_{\text{GT}}\|^2$ and taking square roots gives the claim.

(c) Extend u_{GT} to a smooth function $\tilde{u}$ on $\mathbb{R}^3$ that vanishes away from Ω (e.g., constant along the normal direction near Ω). Then, rescale the coordinates so that $\tilde{u}$ vanishes outside $(-\pi, \pi)^3$ (this is the same as rescaling the weights of the first layer).

Using Fourier expansion, write

$$\tilde{u}(\mathbf{x}) = \sum_{\boldsymbol{\omega} \in \mathbb{Z}^3} c_{\boldsymbol{\omega}}\, e^{i\boldsymbol{\omega}\cdot\mathbf{x}}, \qquad c_{\boldsymbol{\omega}} = \frac{1}{(2\pi)^3} \int_{(-\pi,\pi)^3} \tilde{u}(\mathbf{x})\, e^{-i\boldsymbol{\omega}\cdot\mathbf{x}}\, d\mathbf{x}.$$

Note that $\tilde{u}$ is smooth. Differentiating the series once or twice multiplies $c_{\boldsymbol{\omega}}$ by components of $\boldsymbol{\omega}$, so these series also converge absolutely and uniformly. Consequently the truncated series

$$S_K(\mathbf{x}) = \sum_{\boldsymbol{\omega} \in \mathbb{Z}^3,\ |\omega_1|, |\omega_2|, |\omega_3| \leq K} c_{\boldsymbol{\omega}}\, e^{i\boldsymbol{\omega}\cdot\mathbf{x}}$$

converges to $\tilde{u}$ as $K \to \infty$, uniformly and together with its first and second derivatives.

Since $\tilde{u}$ is real,

$$S_K(\mathbf{x}) = a_{\mathbf{0}} + \sum_{j=1}^{N_K} a_j \sin(\boldsymbol{\omega}_j \cdot \mathbf{x} + b_j), \tag{23}$$

where the sum contains one frequency $\boldsymbol{\omega}_j$ from each pair $\pm\boldsymbol{\omega} \neq \mathbf{0}$ and $N_K = ((2K+1)^3 - 1)/2$. This is exactly a network with one hidden layer of N_K sine neurons: the first-layer weights are $\boldsymbol{\omega}_j$ and the biases are b_j, the output weights are a_j, and the output bias is $a_{\mathbf{0}}$.

The surface laplacian operator in equation 6 is linear and involves only first and second derivatives, with coefficients that depend on $\mathbf{n}$ and are bounded on Ω. Since $\mathcal{L}(\tilde{u}) = 0$ on Ω, $\mathcal{L}(S_K) \to 0$ on Ω as $K \to \infty$.

Our PDE networks have additional hidden layers. These can reproduce the one-hidden-layer network S_K arbitrarily well at the same width N_K. Give the first additional layer the weight matrix $\varepsilon\mathbf{I}$, the later ones the identity matrix, all zero biases, and multiply the output weights by $1/\varepsilon$. For small inputs, $\sin(y) \approx y$, and $\sin(\varepsilon z)/\varepsilon \to z$ as $\varepsilon \to 0$, together with its first and second derivatives, uniformly for $|z| \leq 1$. Hence the deeper network converges to S_K together with its first and second derivatives, and its loss converges to $\mathcal{L}(S_K)$. It follows that $\mathcal{L}^\star_{N_K} \leq \mathcal{L}(S_K)$.

Since $\mathcal{L}(S_K) \to 0$, as $K \to \infty$, and $\mathcal{L}^\star_{N_K} \leq \mathcal{L}(S_K)$, $\mathcal{L}^\star_{N_K} \to 0$ as $K \to \infty$. For any width $N \geq N_K$, statement (a) gives $0 \leq \mathcal{L}^\star_N \leq \mathcal{L}^\star_{N_K}$. Hence $\mathcal{L}^\star_N \to 0$ as $N \to \infty$.

□

Remark. Proposition 1 separates the effect of the network size from that of the optimization. The contribution of the network size, $\mathcal{L}_N^\star$, can only decrease with the width and vanishes in the limit. This matches the error decay in our experiments. The gap η_N depends on the optimization and need not decrease monotonically. Therefore, we observe fluctuations in plots in Fig. 4 and in worst case observe no convergence.
The argument also indicates why high-frequency solutions are harder (Section F): more Fourier modes, and hence a wider network, are needed before $\mathcal{L}_N^\star$ becomes small.

Our current analysis does not provide a convergence rate. It also does not consider soft boundary constraints, error in learnt normals and doesn't bound η_N. We leave a full analysis considering all the cases as future work direction.